REPORT

Diabetic Medication & Patient Re-admission Prediction using different Classification Algorithms

Python Jupyter Framework

Table of Contents

1. Introduction

The management of sugar level in the hospitalized patient has a significant bearing on outcome, in terms of both morbidity and mortality. This recognition has led to the development of formalized protocols in the intensive care unit (ICU) setting with rigorous glucose targets in many institutions. However, the same cannot be said for most non-ICU inpatient admissions. This analysis of a large clinical database is to be undertaken to examine historical patterns of diabetes care in patients admitted to a US hospital and to inform future directions which might lead to improvements in patient's medical condition and help to save medical resources and valuable time of medical staff. Our application will provide general idea of patient's medical necessity based on diabetic prescription and number of times patient is re-admitted in hospital. Since, there has always been shortage of resources in medical industry whether it is hospital bed, medicines and other equipment due to large number of patients, so it is always advisable to keep the tab of patients that might use the resources in future. Our model will provide the predictions in terms of whether patient will be prescribed diabetic medication or not can really help the doctors and medical staff to keep the proper tab of patient's health and avoid diabetic condition. This model will also determine if patient needs to be readmitted in hospital based on clinical history, medication and other factors which can help the medical staff to properly maintain and utilize medical resources as per the necessity and availability.

2. Data

The dataset represents 10 years (1999-2008) of clinical care at 130 US hospitals and integrated delivery networks. It is sourced from UCI Machine Learning Repository and has been prepared to analyze factors related to readmission as well as other outcomes pertaining to patients with diabetes.
(http://archive.ics.uci.edu/ml/datasets/Diabetes+130-US+hospitals+for+years+19992008).

The data contains more than 1,00,000 instances and 50 attributes. The dataset is multivariate in terms of its characteristics, whereas the attributes are numerical and nominal.

Attribute list and their datatype (Numerical or Nominal)

encounter_id : numerical
patient_nbr : numerical
race : nominal
gender : nominal
age : nominal
weight : nominal
admission_type_id : numerical
discharge_disposition_id : numerical
admission_source_id : numerical
time_in_hospital : numerical
payer_code : nominal

medical_speciality : nominal
num_lab_procedures : numerical
num_medications : numerical
number_outpatient : numerical
number_emergency : numerical
numbert_inpatient : numerical
diag_1 : nominal
diag_2 : nominal
diag_3 : nominal
number_diagnoses : numerical
max_glu_serum : nominal
A1Cresult : nominal
metformin : nominal
repaglinide : nominal
nateglinide : nominal
chlorpropamide : nominal
glimepiride : nominal
acetohexamide : nominal
glipizide : nominal
glyburide : nominal
tolbutamide : nominal
pioglitazone : nominal
rosiglitazone : nominal
acarbose : nominal
miglitol : nominal
troglitazone : nominal
tolazamide : nominal
examide : nominal
citoglipton : nominal
insulin : nominal
glyburide-metformin : nominal
glipizide-metformin : nominal
glimepiride-metformin : nominal
metformin-rosiglitazone : nominal
metformin-pioglitazone : nominal
change : nominal
diabetesMed : nominal
readmitted : nominal

3. Problems to be Solved

Based on the above dataset, following are some of the interesting research problems:

1. Based on the clinical condition, patient's physical features and medical history of patient, we are going to predict whether the patient should be treated with diabetic's medication or not. This can be helpful for both patients and doctors as they can keep the tab of the health of their patients with this prediction model and take some early steps to avoid the condition of diabetic medication.

2. In order to improve patient's safety and doctor's valuable time, we are going to determine whether the patient will be required to readmit in the hospital within 30 days, after 30 days or never, based on the clinical history, diabetics prescription and other factors. This can provide the proper timeline of patient's medical history and will help the clinics and hospitals to utilize proper medical resources based on availability of patient and the critical condition of patient, thus avoiding the chaos.

4. KDD

Knowledge Discovery in Databases is the nontrivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data. Following are the basic steps in KDD which were followed to accomplish the goals.

4.1. Data Processing

Basic operations such as the removal of noise if appropriate, collecting the necessary information to model or account for noise, deciding on strategies for handling missing data fields, accounting for time sequence information and known changes which were performed are briefly explained below.

1. Data Cleaning

Data cleansing or data cleaning is the process of detecting and correcting corrupt or inaccurate records from a record set, table, or database and refers to identifying incomplete, incorrect, inaccurate or irrelevant parts of the data and then replacing, modifying, or deleting the dirty or coarse data. Following are the major steps taken to overcome the dirty data.

A. Unique Values

Two columns named "encounter_id" and "patient_number" have unique values for each record mentioned. These columns clearly have no significance in the analysis process and hence were not considered in the further tasks.

B. Single Values

"Examide" and "Citoglipton" consists of same value for all the entries in the dataset. These columns with single value for every possible situation were removed as they do not contribute in analyzing the dataset.

C. *Missing Values*

Columns such as “Weight”, “Medical_Speciality”, “Payer_code” have missing values of 96.9%, 49.1% and 36.6% respectively. Such huge number of null values cannot be replaced therefore, it was finalized to remove these columns.

On the other hand, columns named “Race”, “Diagnosis_1”, “Diagnosis_2” and “Diagnosis_3” had significantly low missing values with 2.2%, 0.02%, 0.35% and 1.39%. These missing values were replaced with the most frequent factor in their respective columns. These were replaced by “Caucasian”, “428”, “276” and “259” respectively.

Checking Missing Values in Dataset:

```
In [7]: # check the null values for each attribute
        data.isnull().sum()

Out[7]: encounter_id                0
        patient_nbr                 0
        race                     2273
        gender                      0
        age                         0
        weight                  98569
        admission_type_id           0
        discharge_disposition_id    0
        admission_source_id         0
        time_in_hospital            0
        payer_code              40256
        medical_specialty       49949
        num_lab_procedures          0
        num_procedures              0
        num_medications             0
        number_outpatient           0
        number_emergency            0
        number_inpatient            0
        diag_1                     21
```

Filling missing values for “Diagnosis_1”, “Diagnosis_2” and “Diagnosis_3”:

```
In [10]: # convert numerical data into string type
         data['diag_1'] = data['diag_1'].astype(str)
         data['diag_2'] = data['diag_2'].astype(str)
         data['diag_3'] = data['diag_3'].astype(str)

In [11]: #fill in the missing values with most frequent term
         data['diag_1'] = data['diag_1'].fillna(data['diag_1'].value_counts().index[0])
         data['diag_2'] = data['diag_2'].fillna(data['diag_2'].value_counts().index[0])
         data['diag_3'] = data['diag_3'].fillna(data['diag_3'].value_counts().index[0])
```

Filling missing values for “Race”:

```
In [18]: #fill in the missing values with most frequent term
         data['race']=data['race'].fillna(data['race'].value_counts().index[0])

In [19]: # check for unique values in race column
         data['race'].unique()

Out[19]: array(['Caucasian', 'AfricanAmerican', 'Other', 'Asian', 'Hispanic'],
               dtype=object)
```

Dropping unnecessary columns:

```
In [21]:  # drop unnecessary columns
          data = data.drop(columns=['encounter_id', 'patient_nbr', 'weight', 'payer_code', 'medical_specialty','examide','citoglipton'])
```

2.Data Integration

Data integration involves combining data residing in different sources and providing users with a unified view of them. This can lead to issues such as schema integration, redundancy, detection and resolution of data value conflicts. To identify and resolve such issues, following measures were performed.

A. Correlation Analysis

Correlation analysis is a statistical method used to evaluate the strength of relationship between two variables. A high correlation means that two or more variables have a strong relationship with each other, while a weak correlation means that the variables are hardly related.

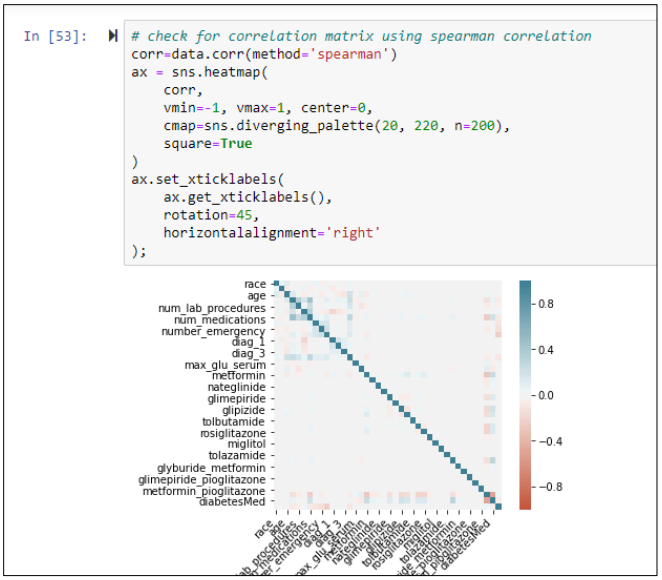
Feedback:

As our targeted variable is binary in nature, we used spearman correlation method. Previously, Pearson Correlation was used.

As shown in the following figure, there no significant issues related to correlations of the variables.

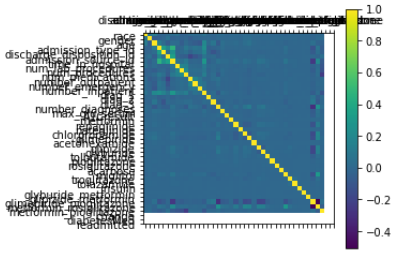
Spearman correlation method:

```
In [53]:  # check for correlation matrix using spearman correlation
          corr=data.corr(method='spearman')
          ax = sns.heatmap(
              corr,
              vmin=-1, vmax=1, center=0,
              cmap=sns.diverging_palette(20, 220, n=200),
              square=True
          )
          ax.set_xticklabels(
              ax.get_xticklabels(),
              rotation=45,
              horizontalalignment='right'
          );
```



Heatmap Correlation:

```
In [51]:
# check for correlation matrix using spearman correlation
import matplotlib
import matplotlib.pyplot as plt
plt.matshow(data.corr(method='spearman'))
plt.xticks(range(len(data.columns)), data.columns)
plt.yticks(range(len(data.columns)), data.columns)
plt.colorbar()
plt.show()
```



B. *Grouping*

Columns named “Diagnosis_1”, “Diagnosis_2” and “Diagnosis_3” have values represented in icd9 codes. In order to make these codes more readable, we converted them into 9 groups as per the pre-defined medical categories.
(https://www.hindawi.com/journals/bmri/2014/781670/tab2/)

Grouping of “Diagnosis_1”, “Diagnosis_2” and “Diagnosis_3”.

```
In [14]:
# create a function to group the records into particular categories
def changeDiag(data, col):
    data[col] = data[col].astype(str)
    for i,x in enumerate(data[col]):
        if(x[0] == 'V' or x[0] == 'E'):
            data.loc[i, col] = "Others"
        else:
            x = float(x)
            if (x >= 250.00 and x < 251):
                data.loc[i, col] = 'Diabetes'
            elif (x >= 390 and x <= 459) or x == 785:
                data.loc[i, col] = 'Circulatory'
            elif (x >= 460 and x<= 519) or x == 786:
                data.loc[i, col] = 'Respiratory'
            elif (x >= 520 and x <= 579) or x == 787:
                data.loc[i, col] = 'Digestive'
            elif (x >= 800 and x <= 999):
                 data.loc[i, col] = 'Injury'
            elif (x >= 710 and x <= 739):
                 data.loc[i, col] = 'Musculoskeletal'
            elif (x >= 580 and x <= 629) or x == 788:
                 data.loc[i, col] = 'Genitourinary'
            elif (x >= 140 and x <= 239):
                 data.loc[i, col] = 'Neoplasms'
            elif (x >= 790 and x <= 799) or x == 780 or x == 781 or x == 784:
                 data.loc[i, col] = 'Others'
            elif (x >= 240 and x <= 279):
```

3.Data Transformation

Data transformation is the process of converting data from one format or structure into another format or structure. The following data transformation methods were performed in order to get the entire dataset in one single form.

A. Data Normalization

The dataset consists of some numerical columns such as “time_in_hospital”, “number_lab_procedures”, “number_medications”, ets. In order to get these numbers in affixed range, Min-Max normalization was performed.

Min-Max normalization:

```
In [22]: # check for numerical columns
         numerics = ['int16', 'int32', 'int64', 'float16', 'float32', 'float64']
         cols_numeric = data.select_dtypes(include=numerics).columns.tolist()

In [23]: cols_numeric

Out[23]: ['time_in_hospital',
          'num_lab_procedures',
          'num_procedures',
          'num_medications',
          'number_outpatient',
          'number_emergency',
          'number_inpatient',
          'number_diagnoses']

In [24]: # use min-max normalization method to scale the numerical columns into particular range
         from sklearn.preprocessing import MinMaxScaler
         scaler = MinMaxScaler()
         data[cols_numeric]=scaler.fit_transform(data[cols_numeric])
```

Data after normalization:

```
In [24]: # use min-max normalization method to scale the numerical columns into particular range
         from sklearn.preprocessing import MinMaxScaler
         scaler = MinMaxScaler()
         data[cols_numeric]=scaler.fit_transform(data[cols_numeric])

In [25]: data.head()

Out[25]:
```

_disposition_id	admission_source_id	time_in_hospital	num_lab_procedures	num_procedures	num_medications	...
25	1	0.000000	0.305344	0.000000	0.0000	...
1	7	0.153846	0.442748	0.000000	0.2125	...
1	7	0.076923	0.076336	0.833333	0.1500	...
1	7	0.076923	0.328244	0.166667	0.1875	...
1	7	0.000000	0.381679	0.000000	0.0875	...

B. *Data Discretization*

Some of the columns such as “Race”, “Gender”, “Age”, etc. with nominal values were required to be converted into numeric data and as a result, LabelEncoder() was used. The pd.cut() was also used to convert numeric data into required groups.

In order to get the dataset in binary form, we converted the numeric data into nominal and created dummy variables.

Data before LableEncoder():

```
In [6]: data.head()
```

Out[6]:

ission_source_id	time_in_hospital	...	citoglipton	insulin	glyburide-metformin	glipizide-metformin	glimepiride-pioglitazone	metformin-rosiglitazone	metformin-pioglitazone	change	diabetesMed	readmitted
1	1	...	No	No	No	No	No	No	No	No	No	NO
7	3	...	No	Up	No	No	No	No	No	Ch	Yes	>30
7	2	...	No	No	No	No	No	No	No	No	Yes	NO
7	2	...	No	Up	No	No	No	No	No	Ch	Yes	NO
7	1	...	No	Steady	No	No	No	No	No	Ch	Yes	NO

LabelEncoder() used:

```
In [26]: # use Label encoding method to provide labels with number
         from sklearn.preprocessing import LabelEncoder

         race_encoder = LabelEncoder()
         data["race"] = race_encoder.fit_transform(data["race"])

In [27]: data['gender'].unique()
Out[27]: array(['Female', 'Male', 'Unknown/Invalid'], dtype=object)

In [28]: gender_encoder = LabelEncoder()
         data["gender"] = gender_encoder.fit_transform(data["gender"])

In [29]: data['age'].unique()
Out[29]: array(['[0-10)', '[10-20)', '[20-30)', '[30-40)', '[40-50)', '[50-60)',
                '[60-70)', '[70-80)', '[80-90)', '[90-100)'], dtype=object)

In [30]: age_encoder = LabelEncoder()
         data["age"] = age_encoder.fit_transform(data["age"])

In [31]: data['diag_3'].unique()
Out[31]: array(['Others', 'Circulatory', 'Diabetes', 'Respiratory', 'Injury',
                'Neoplasms', 'Genitourinary', 'Musculoskeletal', 'Digestive'],
               dtype=object)
```

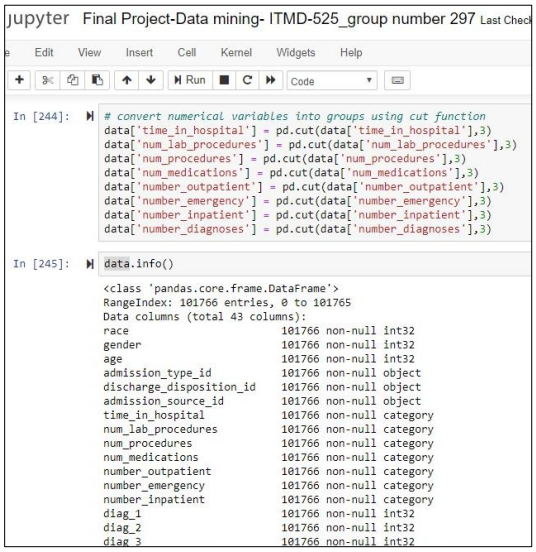
Data After LabelEncoder():

```
In [41]: data.head()
```

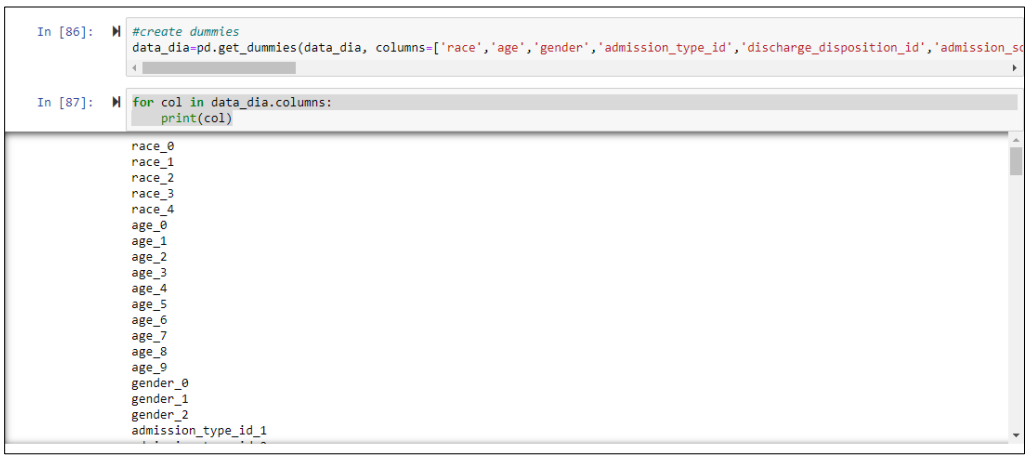
Out[41]:

ns	...	tolazamide	insulin	glyburide_metformin	glipizide_metformin	glimepiride_pioglitazone	metformin_rosiglitazone	metformin_pioglitazone	change	diab
00	...	0	1	1	0	0	0	0	1	
25	...	0	3	1	0	0	0	0	0	
00	...	0	1	1	0	0	0	0	1	
75	...	0	3	1	0	0	0	0	0	
75	...	0	2	1	0	0	0	0	0	

pd.cut():



Dummy Variables:



4.2. Data Splitting

Data splitting is the act of partitioning available data into. two portions, usually for cross-validating purposes. One. portion of the data is used to develop a predictive model. and the other to evaluate the model's performance. As the dataset has more 1,00,000 instances, we used hold-out evaluation to split the data into train and test set in 8:2 ratio.

Data Splitting:

```
In [93]: # split the data into train and test for diabetesMed predictin
         from sklearn.model_selection import train_test_split
         x1_train, x1_test, y1_train, y1_test = train_test_split(data_dia, y_dia, test_size=0.2)
```

4.3. Treating Imbalance Data

To ensure better performance, we checked for imbalance data with respect to both target variables.

Feedback:
We applied SMOTE technique on training data after hold-out evaluation to obtain data balancing. Previously, data balancing was performed on whole dataset before splitting.

A. Predicting Re-admissions

The data with respect to variable “Re-admissions” was distributed in 53.92%, 34.93% and 11.96%. After applying SMOTE technique for over sampling, the improved distribution obtained is 49.7%, 32.2% and 18.11%.

Imbalanced data:

```
In [121]: # split the data into train and test for readmitted prediction
          x2_train, x2_test, y2_train, y2_test = train_test_split(data_re, y_re, test_size=0.2)

In [122]: y2_train.value_counts()

Out[122]: 2    43934
          1    28395
          0     9083
          Name: readmitted, dtype: int64
```

Data after oversampling:

```
In [125]: # apply over-sampling SMOTE technique for imbalanced data in case of readmitted prediction on training dataset
          smt = SMOTE(sampling_strategy={0:15000})
          x2_data, y2_data = smt.fit_sample(x2_train, y2_train)

In [126]: y2_data.value_counts()

Out[126]: 2    43934
          1    28395
          0    15000
          Name: readmitted, dtype: int64
```

B. Predicting Diabetic's Medication Requirement

The data with respect to Diabetic's Medication Requirement was distributed in 22.99% and 77.01%. After applying SMOTE technique for over sampling, the improved distribution obtained is 35.48% and 64.52%.

Imbalanced data:

```
In [115]: y1_train.value_counts()
Out[115]: 1    62636
          0    18776
          Name: diabetesMed, dtype: int64
```

Data after oversampling:

```
In [114]: # apply over-sampling SMOTE technique for imbalanced data in case of diabetesMed prediction on training data set
          from imblearn.over_sampling import SMOTE
          smt = SMOTE(sampling_strategy=0.55)
          x_data, y_data = smt.fit_sample(x1_train, y1_train)

In [116]: y_data.value_counts()
Out[116]: 1    62636
          0    34449
          Name: diabetesMed, dtype: int64
```

4.3. Data Mining Methods and Processes

Classification is the most common task involved in data mining. To solve the proposed research problems, various classification tasks were performed.

A. Predicting Re-admission

The following classification tasks were performed to predict the re-admissions in the hospital based on patient's data.

1. Naïve Bayes

Naive Bayes is a simple technique for constructing classifiers: models that assign class labels to problem instances, represented as vectors of feature values, where the class labels are drawn from some finite set.

After initially applying Naïve Bayes classifier, accuracy of 42% was obtained along with 0.43 precision and 0.42 recall.

Naïve Bayes before balancing data:

```
In [29]:
# readmitted data split
# Naive Bayes classification for readmitted

x2_train, x2_test, y2_train, y2_test = train_test_split(data_final, y2, test_size=0.2)
clf = GaussianNB()
clf.fit(x2_train, y2_train)
y2_pred=clf.predict(x2_test)
print("Accuracy by Hold-out Eval:",accuracy_score(y2_pred,y2_test))
confusion_matrix(y2_test, y2_pred, labels=[0, 1, 2])
precision = precision_score(y2_test, y2_pred, labels=[0,1,2], average='micro')
print('Precision: %.3f' % precision)
recall = recall_score(y2_test, y2_pred, labels=[0,1,2], average='micro')
print('Recall: %.3f' % recall)
f1 = f1_score(y2_test, y2_pred, labels=[0,1,2], average='micro')
print('F1 score: %f' % f1)

Accuracy by Hold-out Eval: 0.14483506667517387
Precision: 0.145
Recall: 0.145
F1 score: 0.144835
```

Accuracy: 0.14 Precision: 0.145 Recall: 0.145

Naïve Bayes after balancing data:

```
In [308]:
# readmitted data split for balanced data
# Naive Bayes classification for readmitted in case of balanced data

x2_train, x2_test, y2_train, y2_test = train_test_split(data_final_re, y2_re, test_size=0.2)

clf = GaussianNB()
clf.fit(x2_train, y2_train)
y2_pred=clf.predict(x2_test)
print("Test Accuracy by Hold-out Eval:",accuracy_score(y2_pred,y2_test))
y2_pred_train=clf.predict(x2_train)
print("Train Accuracy by Hold-out Eval:",accuracy_score(y2_pred_train,y2_train))
confusion_matrix(y2_test, y2_pred, labels=[0, 1, 2])
precision = precision_score(y2_test, y2_pred, labels=[0,1,2], average='micro')
print('Precision: %.3f' % precision)
recall = recall_score(y2_test, y2_pred, labels=[0,1,2], average='micro')
print('Recall: %.3f' % recall)
f1 = f1_score(y2_test, y2_pred, labels=[0,1,2], average='micro')
print('F1 score: %f' % f1)

Test Accuracy by Hold-out Eval: 0.4233824144177621
Train Accuracy by Hold-out Eval: 0.42803892424172046
Precision: 0.423
Recall: 0.423
F1 score: 0.423382
```

Accuracy: 0.42 Precision: 0.43 Recall: 0.42

To further improve the model, PCA technique was used to get top 20 significant variables.

PCA applied:

```
In [309]:
# PCA for readmitted in case of balanced data

from sklearn.decomposition import PCA
from IPython.display import display, HTML

pca = PCA(n_components=20)
fit = pca.fit(data_final_re)

print('Explained variance: ', fit.explained_variance_ratio_)
print('\nPCAs:\n', fit.components_)

PCAs = pca.fit_transform(data_final_re)

# finding top 20 pca components
imp_features = []
for i in range(pca.n_components):
    index = np.where(pca.components_[i] == pca.components_[i].max())
    imp_features.append(index[0][0])

print(data_final_re.iloc[:,imp_features].columns)

x = data_final_re.iloc[:,imp_features]
PCAs.shape
```

```
Explained variance:  [0.04923826 0.02253759 0.02160727 0.02075588 0.01988336 0.01948455
 0.01903317 0.01887763 0.01814429 0.01805346 0.01781488 0.01730511
 0.01707127 0.01671982 0.01640093 0.01614749 0.01605744 0.01577874
 0.01546399 0.01540312]

PCAs:
 [[ 0.08299431  0.01487481  0.02033137 ...  0.00439872  0.00753131
   0.03755342]
 [-0.0306382  -0.01152528 -0.00262131 ... -0.00117136  0.00305161
  -0.00548837]
 [-0.02992027  0.00318566  0.02026131 ...  0.00162629 -0.00235729
   0.00324852]
 ...
 [-0.06749171  0.01373169 -0.004468   ... -0.00024765 -0.0012616
   0.02281482]
 [ 0.01544325 -0.00614809 -0.00589183 ...  0.00810234  0.00241381
   0.00649564]
 [-0.06298108  0.00382704  0.01272912 ... -0.00309019  0.00090491
   0.02234385]]
Index(['diag_3_7', 'diag_3_7', 'diag_2_7', 'diag_3_1',
       'time_in_hospital_(0.333, 0.667]', 'time_in_hospital_(0.333, 0.667]',
       'discharge_disposition_id_6', 'age_7', 'age_5', 'age_6', 'age_6',
       'num_lab_procedures_(0.333, 0.667]', 'admission_type_id_2', 'diag_2_1',
       'diag_3_1', 'age_4', 'num_procedures_(0.333, 0.667]', 'diag_2_0',
       'diag_2_3', 'gender_0'],
      dtype='object')

Out[309]: (107645, 20)
```

Naïve Bayes after PCA:

```
In [310]:
# Naive Bayes after PCA for readmitted in case of balanced data

x2_train, x2_test, y2_train, y2_test = train_test_split(PCAs, y2_re, test_size=0.2)
clf = GaussianNB()
clf.fit(x2_train, y2_train)
y2_pred=clf.predict(x2_test)
print("Accuracy by Hold-out Eval:",accuracy_score(y2_pred,y2_test))
confusion_matrix(y2_test, y2_pred, labels=[0, 1, 2])
precision = precision_score(y2_test, y2_pred, labels=[0,1,2], average='micro')
print('Precision: %.3f' % precision)
recall = recall_score(y2_test, y2_pred, labels=[0,1,2], average='micro')
print('Recall: %.3f' % recall)
f1 = f1_score(y2_test, y2_pred, labels=[0,1,2], average='micro')
print('F1 score: %f' % f1)
```

```
Accuracy by Hold-out Eval: 0.5655627293418181
Precision: 0.566
Recall: 0.566
F1 score: 0.565563
```

Accuracy: 0.56 Precision: 0.566 Recall: 0.566

After balancing the data and performing PCA, the accuracy, precision and recall for the model improved. Thus, the best algorithm is obtained with highest accuracy of 0.56.

2.Decision Tree

A decision tree is a simple representation for classifying examples. Decision tree learning is a method commonly used in data mining. The goal is to create a model that predicts the value of a target variable based on several input variables. The accuracy obtained after initially applying decision tree was 48%. To further improve the results, we tried applying PCA for top 20 significant variables. As a result, we obtained accuracy of 46%.

Decision Tree:

```
In [311]: # decision tree classifier for readmitted in case of balanced data

x2_train, x2_test, y2_train, y2_test = train_test_split(data_final_re, y2_re, test_size=0.2)
clf=DecisionTreeClassifier()
clf=clf.fit(x2_train, y2_train)
y2_pred=clf.predict(x2_test)
acc=accuracy_score(y2_pred, y2_test)
print('Tree Accuracy by hold-out evaluation: ',acc)
confusion_matrix(y2_test, y2_pred, labels=[0, 1, 2])
precision = precision_score(y2_test, y2_pred, labels=[0,1,2], average='micro')
print('Precision: %.3f' % precision)
recall = recall_score(y2_test, y2_pred, labels=[0,1,2], average='micro')
print('Recall: %.3f' % recall)
f1 = f1_score(y2_test, y2_pred, labels=[0,1,2], average='micro')
print('F1 score: %f' % f1)

Tree Accuracy by hold-out evaluation:  0.48836453156207904
Precision: 0.488
Recall: 0.488
F1 score: 0.488365
```

Accuracy: 0.48 Precision: 0.48 Recall: 0.48

Decision tree after PCA (top 20):

```
In [314]: # decision tree classifier for readmitted after PCA (top 20)

x2_train, x2_test, y2_train, y2_test = train_test_split(PCAs, y2_re, test_size=0.2)
clf=DecisionTreeClassifier()
clf=clf.fit(x2_train, y2_train)
y2_pred=clf.predict(x2_test)
print("Accuracy by Hold-out Eval:",accuracy_score(y2_pred,y2_test))
#y2_pred_train=clf.predict(x2_train)
#print("Train Accuracy by Hold-out Eval:",accuracy_score(y2_pred_train,y2_train))
confusion_matrix(y2_test, y2_pred, labels=[0, 1, 2])
precision = precision_score(y2_test, y2_pred, labels=[0,1,2], average='micro')
print('Precision: %.3f' % precision)
recall = recall_score(y2_test, y2_pred, labels=[0,1,2], average='micro')
print('Recall: %.3f' % recall)
f1 = f1_score(y2_test, y2_pred, labels=[0,1,2], average='micro')
print('F1 score: %f' % f1)

Accuracy by Hold-out Eval: 0.46147057457383067
Precision: 0.461
Recall: 0.461
F1 score: 0.461471
```

Accuracy: 0.46 Precision: 0.46 Recall: 0.46

Even after performing PCA for top 20 variables, the accuracy was not improved. Thus the highest accuracy obtained is 0.48.

3. Bagging- Decision Tree Classifier

Bootstrap aggregating, also called bagging (from bootstrap aggregating), is a machine learning ensemble meta-algorithm designed to improve the stability and accuracy of machine learning algorithms used in statistical classification and regression. It also reduces variance and helps to avoid overfitting. Although it is usually applied to decision tree methods, it can be used with any type of method. Bagging is a special case of the model averaging approach.

Feedback:

We applied bagging algorithm to the decision tree classifier to get better accuracy. Previously, the classifier used for bagging was not mentioned.

Initial accuracy obtained with bagging was 55%. After applying PCA for top 20 variables, we got an accuracy of 55% as well.

Bagging – Decision Tree Classifier :

```
In [315]: # Bagging method for readmission using Decision Tree classifier

x2_train, x2_test, y2_train, y2_test = train_test_split(data_final_re, y2_re, test_size=0.2)
tree = DecisionTreeClassifier()
bag = BaggingClassifier(tree, n_estimators=100, max_samples=0.8, random_state=1)
bag=bag.fit(x2_train, y2_train)
y2_pred=bag.predict(x2_test)
acc=accuracy_score(y2_pred, y2_test)
print('Tree Accuracy by hold-out evaluation: ',acc)
confusion_matrix(y2_test, y2_pred, labels=[0, 1, 2])
precision = precision_score(y2_test, y2_pred, labels=[0,1,2], average='micro')
print('Precision: %.3f' % precision)
recall = recall_score(y2_test, y2_pred, labels=[0,1,2], average='micro')
print('Recall: %.3f' % recall)
f1 = f1_score(y2_test, y2_pred, labels=[0,1,2], average='micro')
print('F1 score: %f' % f1)

Tree Accuracy by hold-out evaluation:  0.559988852245808
Precision: 0.560
Recall: 0.560
F1 score: 0.559989
```

Accuracy: 0.55 Precision: 0.56 Recall: 0.56

Bagging – Decision Tree Classifier: After PCA:

```
In [324]: #  Bagging method for readmission using Decision Tree classifier after PCA

x2_train, x2_test, y2_train, y2_test = train_test_split(PCAs, y2_re, test_size=0.2)
tree = DecisionTreeClassifier()
bag = BaggingClassifier(tree, n_estimators=128, max_samples=0.8, random_state=1)
bag=bag.fit(x2_train, y2_train)
y2_pred=bag.predict(x2_test)
acc=accuracy_score(y2_pred, y2_test)
print('Tree Accuracy by hold-out evaluation: ',acc)
confusion_matrix(y2_test, y2_pred, labels=[0, 1, 2])
precision = precision_score(y2_test, y2_pred, labels=[0,1,2], average='micro')
print('Precision: %.3f' % precision)
recall = recall_score(y2_test, y2_pred, labels=[0,1,2], average='micro')
print('Recall: %.3f' % recall)
f1 = f1_score(y2_test, y2_pred, labels=[0,1,2], average='micro')
print('F1 score: %f' % f1)

Tree Accuracy by hold-out evaluation:  0.5504668122067908
Precision: 0.550
Recall: 0.550
F1 score: 0.550467
```

Accuracy: 0.55 Precision: 0.55 Recall: 0.55

Thus, the best working algorithm has accuracy of 0.55, precision of 0.56 and recall of 0.56

4. Random Forest

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes or mean prediction of the individual trees.

We tried applying Random Forest algorithm to observe the results before and after applying PCA for top 20 variables. And thus, got the following accuracies:

Random Forest:

```
In [316]: # random forest classifier for readmitted

          x2_train, x2_test, y2_train, y2_test = train_test_split(data_final_re, y2_re, test_size=0.2)

          from sklearn.ensemble import RandomForestClassifier
          from sklearn.metrics import roc_auc_score
          model = RandomForestClassifier(n_estimators=128,
                                         bootstrap = True,
                                         max_features = 'sqrt')
          model.fit(x2_train, y2_train)
          model.feature_importances_
          rf_predictions = model.predict(x2_test)
          rf_probs = model.predict_proba(x2_test)
          roc_value = roc_auc_score(y2_test, rf_probs,multi_class='ovr')
```

```
In [317]: print(roc_value)
          print(rf_probs)
          print(rf_predictions)

          0.6809808762009548
          [[0.0625     0.2265625  0.7109375 ]
           [0.1171875  0.453125   0.4296875 ]
           [0.078125   0.7734375  0.1484375 ]
           ...
           [0.1484375  0.51953125 0.33203125]
           [0.2421875  0.390625   0.3671875 ]
           [0.1328125  0.3671875  0.5       ]]
          [2 1 1 ... 1 1 2]
```

```
In [271]: print("Accuracy by Hold-out Eval:",accuracy_score(rf_predictions,y2_test))
          confusion_matrix(y2_test, rf_predictions)
          precision = precision_score(y2_test, rf_predictions, average='micro')
          print('Precision: %.3f' % precision)
          recall = recall_score(y2_test, rf_predictions, average='micro')
          print('Recall: %.3f' % recall)
          f1 = f1_score(y2_test, rf_predictions, average='micro')
          print('F1 score: %f' % f1)

          Accuracy by Hold-out Eval: 0.5731692257028774
          Precision: 0.573
          Recall: 0.573
          F1 score: 0.573169
```

Accuracy: 0.57 Precision: 0.53 Recall: 0.57

Random Forest after PCA:

```
In [325]:
# random forest classifier for readmitted after PCA

x2_train, x2_test, y2_train, y2_test = train_test_split(PCAs, y2_re, test_size=0.2)

from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_auc_score
model = RandomForestClassifier(n_estimators=100,
                               bootstrap = True,
                               max_features = 'sqrt')
model.fit(x2_train, y2_train)
model.feature_importances_
rf_predictions = model.predict(x2_test)
rf_probs = model.predict_proba(x2_test)
roc_value = roc_auc_score(y2_test, rf_probs,multi_class='ovr')
```

```
In [326]:
print(roc_value)
print(rf_probs)
print(rf_predictions)
print("Accuracy by Hold-out Eval:",accuracy_score(rf_predictions,y2_test))
confusion_matrix(y2_test, rf_predictions)
precision = precision_score(y2_test, rf_predictions, average='micro')
print('Precision: %.3f' % precision)
recall = recall_score(y2_test, rf_predictions, average='micro')
print('Recall: %.3f' % recall)
f1 = f1_score(y2_test, rf_predictions, average='micro')
print('F1 score: %f' % f1)
```

```
In [326]:
print(roc_value)
print(rf_probs)
print(rf_predictions)
print("Accuracy by Hold-out Eval:",accuracy_score(rf_predictions,y2_test))
confusion_matrix(y2_test, rf_predictions)
precision = precision_score(y2_test, rf_predictions, average='micro')
print('Precision: %.3f' % precision)
recall = recall_score(y2_test, rf_predictions, average='micro')
print('Recall: %.3f' % recall)
f1 = f1_score(y2_test, rf_predictions, average='micro')
print('F1 score: %f' % f1)

0.6404143253531931
[[0.1   0.2   0.7  ]
 [0.155 0.495 0.35 ]
 [0.11  0.47  0.42 ]
 ...
 [1.    0.    0.   ]
 [0.09  0.33  0.58 ]
 [0.145 0.445 0.41 ]]
[2 1 1 ... 0 2 1]
Accuracy by Hold-out Eval: 0.548887547029588
Precision: 0.549
Recall: 0.549
F1 score: 0.548888
```

Accuracy: 0.54 Precision: 0.54 Recall: 0.54

As a result, the highest accuracy obtained is 0.57.

5. Gradient Boosting- Decision Tree Classifier

Gradient boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees.

Feedback:

We used gradient boosting method for decision tree classifier and observed the outcomes. Previously, the name of the algorithm on which gradient boosting is applied was not mentioned.

Gradient Boosting was also applied to the pre-processed data to observe the behaviors. Initially we obtained an accuracy of 58% but after applying PCA for top 20 most significant variables, it was changed to 54%.

Gradient Boost- Decision Tree Classifier:

```
In [*]: # gradient boosting using ensemble for readmitted - decision tree classifier

x2_train, x2_test, y2_train, y2_test = train_test_split(data_final_re, y2_re, test_size=0.2)

params = {'n_estimators': 128, 'loss':'deviance', 'max_depth': 16, 'min_samples_split': 64,
          'learning_rate': 0.1,'max_features':'sqrt','verbose':4}
clf = ensemble.GradientBoostingClassifier(**params)
clf = clf.fit(x2_train, y2_train)
y2_pred=clf.predict(x2_test)

      Iter       Train Loss   Remaining Time
         1       84668.0737           1.22m
         2       83382.9911           1.21m
         3       82271.0502           1.19m
         4       81376.9220           1.15m
         5       80563.5551           1.15m
         6       79855.3831           1.15m
         7       79207.2393           1.15m
         8       78626.2822           1.14m
         9       78071.6882           1.14m
        10       77508.4486           1.15m
        11       77064.7447           1.13m
        12       76648.1568           1.13m
        13       76274.0744           1.12m
        14       75821.1821           1.12m
        15       75425.1612           1.12m
        16       75111.8020           1.10m
       118       48452.9760          23.72s
       119       48333.7302          21.36s
       120       48144.6797          19.00s
       121       47900.6922          16.62s
       122       47748.6166          14.24s
       123       47385.5574          11.86s
       124       47092.2877           9.49s
       125       46952.0863           7.12s
       126       46777.3977           4.75s
       127       46606.6907           2.37s
       128       46495.3421           0.00s

In [319]: acc=accuracy_score(y2_pred, y2_test)
          print('Tree Accuracy by hold-out evaluation: ',acc)
          confusion_matrix(y2_test, y2_pred, labels=[0, 1, 2])
          precision = precision_score(y2_test, y2_pred, labels=[0,1,2], average='micro')
          print('Precision: %.3f' % precision)
          recall = recall_score(y2_test, y2_pred, labels=[0,1,2], average='micro')
          print('Recall: %.3f' % recall)
          f1 = f1_score(y2_test, y2_pred, labels=[0,1,2], average='micro')
          print('F1 score: %f' % f1)

          Tree Accuracy by hold-out evaluation:  0.581680523944447
          Precision: 0.582
          Recall: 0.582
          F1 score: 0.581681
```

Accuracy: 0.58 Precision: 0.58 Recall: 0.58

Gradient Boost- Decision Tree Classifier after PCA:

```
In [327]: # gradient boosting using ensemble-decision tree classifier for readmitted after PCA

x2_train, x2_test, y2_train, y2_test = train_test_split(PCAs, y2_re, test_size=0.2)

params = {'n_estimators': 128, 'loss':'deviance', 'max_depth': 16, 'min_samples_split': 64,
          'learning_rate': 0.1,'max_features':'sqrt','verbose':4}
clf = ensemble.GradientBoostingClassifier(**params)
clf = clf.fit(x2_train, y2_train)
y2_pred=clf.predict(x2_test)
```

```
      110       27896.4697           35.93s
      111       27645.6746           33.94s
      112       27430.6886           31.94s
      113       27157.4217           29.95s
```

```
      124       24750.7040            8.07s
      125       24579.3092            6.06s
      126       24352.0263            4.04s
      127       24157.1689            2.02s
      128       23972.1893            0.00s
```

```
In [328]: acc=accuracy_score(y2_pred, y2_test)
print('Tree Accuracy by hold-out evaluation: ',acc)
confusion_matrix(y2_test, y2_pred, labels=[0, 1, 2])
precision = precision_score(y2_test, y2_pred, labels=[0,1,2], average='micro')
print('Precision: %.3f' % precision)
recall = recall_score(y2_test, y2_pred, labels=[0,1,2], average='micro')
print('Recall: %.3f' % recall)
f1 = f1_score(y2_test, y2_pred, labels=[0,1,2], average='micro')
print('F1 score: %f' % f1)

Tree Accuracy by hold-out evaluation:  0.5484230572715871
Precision: 0.548
Recall: 0.548
F1 score: 0.548423
```

Accuracy: 0.54 Precision: 0.54 Recall: 0.54

The best working algorithm using gradient boost for decision tree classifier has an accuracy of 0.58.

6. SVM- Support Vector Machine

In machine learning, support-vector machines are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis.

Lastly, SVM was performed and following results were obtained.

SVM - Support Vector Machine:

```
In [320]:
# SVM for readmitted

from sklearn.svm import LinearSVC

# by hold-out evaluation
x2_train, x2_test, y2_train, y2_test = train_test_split(data_final_re, y2_re, test_size=0.2)

#clf=SVC(kernel='linear', C=1E10) # C is large -> hard margin; C is small -> soft margin
clf = LinearSVC(random_state=0, tol=1e-5)
clf=clf.fit(x2_train, y2_train)
y2_pred=clf.predict(x2_test)
acc=accuracy_score(y2_pred, y2_test)
print('Tree Accuracy by hold-out evaluation: ',acc)
confusion_matrix(y2_test, y2_pred, labels=[0, 1, 2])
precision = precision_score(y2_test, y2_pred, labels=[0,1,2], average='micro')
print('Precision: %.3f' % precision)
recall = recall_score(y2_test, y2_pred, labels=[0,1,2], average='micro')
print('Recall: %.3f' % recall)
f1 = f1_score(y2_test, y2_pred, labels=[0,1,2], average='micro')
print('F1 score: %f' % f1)

Tree Accuracy by hold-out evaluation:  0.5795903200334432
Precision: 0.580
Recall: 0.580
F1 score: 0.579590
```

Accuracy: 0.57 Precision: 0.58 Recall: 0.58

SVM- Support Vector Machine after PCA:

```
In [329]:
# SVM for readmitted after PCA

from sklearn.svm import LinearSVC

# by hold-out evaluation
x2_train, x2_test, y2_train, y2_test = train_test_split(PCAs, y2_re, test_size=0.2)

#clf=SVC(kernel='linear', C=1E10) # C is large -> hard margin; C is small -> soft margin
clf = LinearSVC(random_state=0, tol=1e-5)
clf=clf.fit(x2_train, y2_train)
y2_pred=clf.predict(x2_test)
acc=accuracy_score(y2_pred, y2_test)
print('Tree Accuracy by hold-out evaluation: ',acc)
confusion_matrix(y2_test, y2_pred, labels=[0, 1, 2])
precision = precision_score(y2_test, y2_pred, labels=[0,1,2], average='micro')
print('Precision: %.3f' % precision)
recall = recall_score(y2_test, y2_pred, labels=[0,1,2], average='micro')
print('Recall: %.3f' % recall)
f1 = f1_score(y2_test, y2_pred, labels=[0,1,2], average='micro')
print('F1 score: %f' % f1)

Tree Accuracy by hold-out evaluation:  0.5616145663988109
Precision: 0.562
Recall: 0.562
F1 score: 0.561615
```

Accuracy: 0.56 Precision: 0.56 Recall: 0.56

Thus, the highest accuracy obtained for SVM is 0.57.

B. Predicting Diabetic's Medication Requirement

The following classification tasks were performed to predict the diabetic's medication requirements for the patients based on their profile.

1. Naïve Bayes

Naive Bayes is a simple technique for constructing classifiers: models that assign class labels to problem instances, represented as vectors of feature values, where the class labels are drawn from some finite set.

After initially applying Naïve Bayes classifier, accuracy of 80% was obtained along with 0.77 precision and 1.00 recall.

Naïve Bayes before balancing data:

```
In [95]: clf = GaussianNB()
         clf.fit(x1_train, y1_train)
         y1_pred=clf.predict(x1_test)
         print("Accuracy by Hold-out Eval:",accuracy_score(y1_pred,y1_test))

         Accuracy by Hold-out Eval: 0.8277488454357865

In [79]: from sklearn.metrics import confusion_matrix

In [96]: confusion_matrix(y1_test, y1_pred)

Out[96]: array([[ 4699,     0],
                [ 3506, 12149]], dtype=int64)

In [97]: precision = precision_score(y1_test, y1_pred, average='binary')

In [98]: print('Precision: %.3f' % precision)

         Precision: 1.000

In [99]: recall = recall_score(y1_test, y1_pred, average='binary')
         print('Recall: %.3f' % recall)

         Recall: 0.776
```

Accuracy: 0.82 Precision: 1 Recall: 0.766

Naïve Bayes after balancing data:

```
In [284]: # diabetesMed data split for balanced data
          # Naive Bayes classification for diabetesMed in case of balanced data

          x1_train, x1_test, y1_train, y1_test = train_test_split(data_final_dia, y1_dia, test_size=0.2)
          clf = GaussianNB()
          clf.fit(x_data, y_data)
          y1_pred=clf.predict(x1_test)
          print("Test Accuracy by Hold-out Eval:",accuracy_score(y1_pred,y1_test))
          y1_pred_train=clf.predict(x1_train)
          print("Train Accuracy by Hold-out Eval:",accuracy_score(y1_pred_train,y1_train))
          confusion_matrix(y1_test, y1_pred)
          precision = precision_score(y1_test, y1_pred, average='binary')
          print('Precision: %.3f' % precision)
          recall = recall_score(y1_test, y1_pred, average='binary')
          print('Recall: %.3f' % recall)
          f1 = f1_score(y1_test, y1_pred, average='binary')
          print('F1 score: %f' % f1)

          Test Accuracy by Hold-out Eval: 0.8063020921925498
          Train Accuracy by Hold-out Eval: 0.8037485515027163
          Precision: 0.776
          Recall: 1.000
          F1 score: 0.873665
```

Accuracy: 0.80 Precision: 0.77 Recall: 1.00

To further improve the model, PCA technique was used to get top 20 significant variables.

PCA applied:

```
In [177]:
# PCA for diabetesMed in case of balanced data

from sklearn.decomposition import PCA
from IPython.display import display, HTML

pca = PCA(n_components=20)
fit = pca.fit(data_final_dia)

print('Explained variance: ', fit.explained_variance_ratio_)
print('\nPCAs:\n', fit.components_)

PCAs = pca.fit_transform(data_final_dia)

# finding top 20 pca components
imp_features = []
for i in range(pca.n_components):
    index = np.where(pca.components_[i] == pca.components_[i].max())
    imp_features.append(index[0][0])

print(data_final_dia.iloc[:,imp_features].columns)

x = data_final_dia.iloc[:,imp_features]
PCAs.shape
```

```
x = data_final_dia.iloc[:,imp_features]
PCAs.shape

Explained variance:  [0.03568953 0.02514494 0.02398341 0.02328773 0.02259714 0.02169853
 0.02136592 0.02065816 0.02025407 0.01981543 0.01942153 0.01905665
 0.0186315  0.01810617 0.01786031 0.01776592 0.01761948 0.01725684
 0.01717093 0.01694496]

PCAs:
 [[ 7.47592071e-02  2.21979036e-02  7.45586447e-03 ...  1.18476258e-01
    3.15436954e-02  1.56916118e-03]
 [-3.67608202e-02 -9.44788045e-03 -2.48103580e-03 ... -1.01270531e-01
    2.54748639e-03 -1.43557579e-03]
 [-4.74157552e-03 -5.26609269e-03 -2.52133146e-03 ... -4.23169085e-02
    5.17880232e-03  1.45016983e-03]
 ...
 [-5.02370923e-02  1.24603987e-02  9.62950598e-03 ...  4.06590595e-01
   -1.83195094e-02  2.16774585e-03]
 [-2.14460467e-03  5.61380177e-03 -2.46776876e-03 ... -8.79095598e-04
    1.11724578e-02  8.70174350e-04]
 [-8.67521127e-03 -9.43946366e-05  1.64423397e-03 ... -2.17014335e-01
   -1.92057236e-02  3.31041551e-03]]
Index(['time_in_hospital_(0.333, 0.667]', 'time_in_hospital_(0.333, 0.667]',
       'age_5', 'diag_2_7', 'age_7', 'diag_1_7',
       'num_procedures_(0.333, 0.667]', 'discharge_disposition_id_6',
       'diag_3_0', 'diag_3_0', 'admission_type_id_2', 'diag_3_7', 'diag_3_1',
       'diag_3_1', 'diag_1_8', 'diag_1_8', 'num_lab_procedures_(0.333, 0.667]',
       'gender_1', 'num_lab_procedures_(0.333, 0.667]',
       'time_in_hospital_(0.667, 1.0]'],
      dtype='object')
```

Naïve Bayes after PCA:

```
In [178]:
# Naive Bayes after PCA for diabetesMed

x1_train, x1_test, y1_train, y1_test = train_test_split(PCAs, y1_dia, test_size=0.2)
clf = GaussianNB()
clf.fit(x1_train, y1_train)
y1_pred=clf.predict(x1_test)
print("Test Accuracy by Hold-out Eval:",accuracy_score(y1_pred,y1_test))
y1_pred_train=clf.predict(x1_train)
print("Train Accuracy by Hold-out Eval:",accuracy_score(y1_pred_train,y1_train))
confusion_matrix(y1_test, y1_pred)
precision = precision_score(y1_test, y1_pred, average='binary')
print('Precision: %.3f' % precision)
recall = recall_score(y1_test, y1_pred, average='binary')
print('Recall: %.3f' % recall)
f1 = f1_score(y1_test, y1_pred, average='binary')
print('F1 score: %f' % f1)

Test Accuracy by Hold-out Eval: 0.7901481607629428
Train Accuracy by Hold-out Eval: 0.7879532948026099
Precision: 0.761
Recall: 1.000
F1 score: 0.864249
```

Accuracy: 0.79 Precision: 0.76 Recall: 1.00

The best working algorithm using Naïve Bayes classifier has accuracy of 80%.

2. Decision Tree

A decision tree is a simple representation for classifying examples. Decision tree learning is a method commonly used in data mining. The goal is to create a model that predicts the value of a target variable based on several input variables. The accuracy obtained after initially applying decision tree was 1.00. Further, we tried applying PCA for top 20 significant variables. As a result, we obtained accuracies as 1.00 and 1.00 respectively.

Decision Tree:

```
In [190]:
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier

In [286]:
# decision tree classifier for diabetesMed in case of balanced data

#x1_train, x1_test, y1_train, y1_test = train_test_split(data_final_dia, y1_dia, test_size=0.2)
clf=DecisionTreeClassifier()
clf=clf.fit(x1_train, y1_train)
y1_pred=clf.predict(x1_test)
print("Test Accuracy by Hold-out Eval:",accuracy_score(y1_pred,y1_test))
y1_pred_train=clf.predict(x1_train)
print("Train Accuracy by Hold-out Eval:",accuracy_score(y1_pred_train,y1_train))
confusion_matrix(y1_test, y1_pred)
precision = precision_score(y1_test, y1_pred, average='binary')
print('Precision: %.3f' % precision)
recall = recall_score(y1_test, y1_pred, average='binary')
print('Recall: %.3f' % recall)
f1 = f1_score(y1_test, y1_pred, average='binary')
print('F1 score: %f' % f1)

Test Accuracy by Hold-out Eval: 1.0
Train Accuracy by Hold-out Eval: 1.0
Precision: 1.000
Recall: 1.000
F1 score: 1.000000
```

Accuracy: 1.00 Precision: 1.00 Recall: 1.00

Decision tree after PCA (top 20):

```
In [300]:  # decision tree classifier for diabetesMed after PCA

x1_train, x1_test, y1_train, y1_test = train_test_split(PCAs, y1_dia, test_size=0.2)
clf=DecisionTreeClassifier()
clf=clf.fit(x1_train, y1_train)
y1_pred=clf.predict(x1_test)
print("Test Accuracy by Hold-out Eval:",accuracy_score(y1_pred,y1_test))
y1_pred_train=clf.predict(x1_train)
print("Train Accuracy by Hold-out Eval:",accuracy_score(y1_pred_train,y1_train))
confusion_matrix(y1_test, y1_pred)
precision = precision_score(y1_test, y1_pred, average='binary')
print('Precision: %.3f' % precision)
recall = recall_score(y1_test, y1_pred, average='binary')
print('Recall: %.3f' % recall)
f1 = f1_score(y1_test, y1_pred, average='binary')
print('F1 score: %f' % f1)

Test Accuracy by Hold-out Eval: 0.716363327096445
Train Accuracy by Hold-out Eval: 1.0
Precision: 0.798
Recall: 0.770
F1 score: 0.783554
```

Accuracy: 1.00 Precision: 0.79 Recall: 0.77

As a result, highest accuracy obtained is 1.

3. Bagging- Decision Tree Classifier

Bootstrap aggregating, also called bagging (from bootstrap aggregating), is a machine learning ensemble meta-algorithm designed to improve the stability and accuracy of machine learning algorithms used in statistical classification and regression. It also reduces variance and helps to avoid overfitting. Although it is usually applied to decision tree methods, it can be used with any type of method. Bagging is a special case of the model averaging approach.

Feedback:

We applied bagging algorithm to the decision tree classifier to get better accuracy. Previously, the classifier used for bagging was not mentioned.

Initial accuracy obtained with bagging was 0.99. After applying PCA for top 20 variables, we got an accuracy of 0.78.

Bagging:

```
In [287]:
# Bagging method for diabetesMed using Decision Tree classifier

#x1_train, x1_test, y1_train, y1_test = train_test_split(data_b_diab, y1_d, test_size=0.2)
tree = DecisionTreeClassifier()
bag = BaggingClassifier(tree, n_estimators=100, max_samples=0.8, random_state=1)
bag=bag.fit(x1_train, y1_train)
y1_pred=bag.predict(x1_test)
print("Test Accuracy by Hold-out Eval:",accuracy_score(y1_pred,y1_test))
y1_pred_train=bag.predict(x1_train)
print("Train Accuracy by Hold-out Eval:",accuracy_score(y1_pred_train,y1_train))
confusion_matrix(y1_test, y1_pred)
precision = precision_score(y1_test, y1_pred, average='binary')
print('Precision: %.3f' % precision)
recall = recall_score(y1_test, y1_pred, average='binary')
print('Recall: %.3f' % recall)
f1 = f1_score(y1_test, y1_pred, average='binary')
print('F1 score: %f' % f1)

Test Accuracy by Hold-out Eval: 0.9999574757611839
Train Accuracy by Hold-out Eval: 1.0
Precision: 1.000
Recall: 1.000
F1 score: 0.999968
```

Accuracy: 0.99 Precision: 1.00 Recall: 1.00

Bagging After PCA:

```
In [302]:
# Bagging method for diabetesMed using Decision Tree classifier after PCA

x1_train, x1_test, y1_train, y1_test = train_test_split(PCAs, y1_dia, test_size=0.2)
tree = DecisionTreeClassifier()
bag = BaggingClassifier(tree, n_estimators=128, max_samples=0.8, random_state=1)
bag=bag.fit(x1_train, y1_train)
y1_pred=bag.predict(x1_test)
acc=accuracy_score(y1_pred, y1_test)
print('Tree Accuracy by hold-out evaluation: ',acc)
confusion_matrix(y1_test, y1_pred)
precision = precision_score(y1_test, y1_pred, average='binary')
print('Precision: %.3f' % precision)
recall = recall_score(y1_test, y1_pred, average='binary')
print('Recall: %.3f' % recall)
f1 = f1_score(y1_test, y1_pred, average='binary')
print('F1 score: %f' % f1)

Tree Accuracy by hold-out evaluation:  0.7895900663378126
Precision: 0.788
Recall: 0.935
F1 score: 0.854948
```

Accuracy: 0.78 Precision: 0.78 Recall: 0.93

4. Random Forest

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes or mean prediction of the individual trees.

We tried applying Random Forest algorithm to observe the results before and after applying PCA for top 20 variables. And thus, got the following accuracies:

Random Forest:

```
In [288]: # random forest classifier for diabetesMed

          #x1_train, x1_test, y1_train, y1_test = train_test_split(data_b_diab, y1_d, test_size=0.2)

          from sklearn.ensemble import RandomForestClassifier
          from sklearn.metrics import roc_auc_score
          model = RandomForestClassifier(n_estimators=100,
                                         bootstrap = True,
                                         max_features = 'sqrt')
          model.fit(x1_train, y1_train)
          model.feature_importances_
          rf_predictions = model.predict(x1_test)
          rf_probs = model.predict_proba(x1_test)[:, 1]
          roc_value = roc_auc_score(y1_test, rf_probs)

In [289]: print(roc_value)
          print(rf_probs)
          print(rf_predictions)

          1.0
          [1. 0. 0. ... 1. 1. 1.]
          [1 0 0 ... 1 1 1]

In [290]: print("Accuracy by Hold-out Eval:",accuracy_score(rf_predictions,y1_test))
          confusion_matrix(y1_test, rf_predictions)
          precision = precision_score(y1_test, rf_predictions, average='binary')
          print('Precision: %.3f' % precision)
          recall = recall_score(y1_test, rf_predictions, average='binary')
          print('Recall: %.3f' % recall)
          f1 = f1_score(y1_test, rf_predictions, average='binary')
          print('F1 score: %f' % f1)

          Accuracy by Hold-out Eval: 0.9999149515223678
          Precision: 1.000
          Recall: 1.000
          F1 score: 0.999937
```

Accuracy: 0.99 Precision: 1.00 Recall: 1.00

Random Forest after PCA:

```
In [303]:
# random forest classifier for diabetesMed after PCA

x1_train, x1_test, y1_train, y1_test = train_test_split(PCAs, y1_dia, test_size=0.2)

from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_auc_score
model = RandomForestClassifier(n_estimators=100,
                               bootstrap = True,
                               max_features = 'sqrt')
model.fit(x1_train, y1_train)
model.feature_importances_
rf_predictions = model.predict(x1_test)
rf_probs = model.predict_proba(x1_test)[:, 1]
roc_value = roc_auc_score(y1_test, rf_probs)
```

```
In [304]:
print(roc_value)
print(rf_probs)
print(rf_predictions)
print("Accuracy by Hold-out Eval:",accuracy_score(rf_predictions,y1_test))
confusion_matrix(y1_test, rf_predictions)
precision = precision_score(y1_test, rf_predictions, average='binary')
print('Precision: %.3f' % precision)
recall = recall_score(y1_test, rf_predictions, average='binary')
print('Recall: %.3f' % recall)
f1 = f1_score(y1_test, rf_predictions, average='binary')
print('F1 score: %f' % f1)
```

```
0.7664867831552328
[0.5  0.74 0.79 ... 0.77 0.68 0.87]
[0 1 1 ... 1 1 1]
Accuracy by Hold-out Eval: 0.7942252083687702
Precision: 0.794
Recall: 0.933
F1 score: 0.858206
```

Accuracy: 0.766 Precision: 0.79 Recall: 0.93

5. Gradient Boost- Decision Tree Classifier

Gradient boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees.

Feedback:

We used gradient boosting method for decision tree classifier and observed the outcomes. Previously, the name of the algorithm on which gradient boosting is applied was not mentioned.

Gradient Boosting was also applied to the pre-processed data to observe the behaviors. Initially we obtained an accuracy of 1.00 but after applying PCA for top 20 most significant variables, it was changed to 0.79.

Gradient Boostng:

```
In [*]: # gradient boosting using ensemble method for diabetesMed- Decision tree

#x1_train, x1_test, y1_train, y1_test = train_test_split(data_b_diab, y1_d, test_size=0.2)

params = {'n_estimators': 100, 'loss':'deviance', 'max_depth': 16, 'min_samples_split': 64,
          'learning_rate': 0.1,'max_features':'sqrt','verbose':4}
clf = ensemble.GradientBoostingClassifier(**params)
clf = clf.fit(x1_train, y1_train)
y1_pred=clf.predict(x1_test)
```

```
      Iter       Train Loss   Remaining Time
         1           1.1027            9.58s
         2           0.9598            9.67s
         3           0.8843           10.13s
         4           0.7980           10.32s
         5           0.7061           11.55s
         6           0.6259           11.94s
         7           0.5836           12.33s
         8           0.5464           12.65s
         9           0.5149           12.86s
        10           0.4600           13.10s
        11           0.4184           13.26s
        12           0.3786           13.28s
        13           0.3468           13.41s
        14           0.3136           13.37s
        15           0.2892           13.31s
        16           0.2613           13.30s
```

```
In [293]: print("Test Accuracy by Hold-out Eval:",accuracy_score(y1_pred,y1_test))
y1_pred_train=clf.predict(x1_train)
print("Train Accuracy by Hold-out Eval:",accuracy_score(y1_pred_train,y1_train))
confusion_matrix(y1_test, y1_pred)
precision = precision_score(y1_test, y1_pred, average='binary')
print('Precision: %.3f' % precision)
recall = recall_score(y1_test, y1_pred, average='binary')
print('Recall: %.3f' % recall)
f1 = f1_score(y1_test, y1_pred, average='binary')
print('F1 score: %f' % f1)
```

```
Test Accuracy by Hold-out Eval: 1.0
Train Accuracy by Hold-out Eval: 1.0
Precision: 1.000
Recall: 1.000
F1 score: 1.000000
```

Accuracy: 1.00 Precision: 1.00 Recall: 1.00

Gradient Boost after PCA:

```
In [*]: # gradient boosting for diabetesMed after PCA

x1_train, x1_test, y1_train, y1_test = train_test_split(PCAs, y1_dia, test_size=0.2)

params = {'n_estimators': 128, 'loss':'deviance', 'max_depth': 16, 'min_samples_split': 64,
          'learning_rate': 0.1,'max_features':'sqrt','verbose':4}
clf = ensemble.GradientBoostingClassifier(**params)
clf = clf.fit(x1_train, y1_train)
y1_pred=clf.predict(x1_test)
```

```
      Iter       Train Loss   Remaining Time
         1           1.2074           1.40m
         2           1.1543           1.36m
         3           1.1127           1.36m
         4           1.0786           1.33m
         5           1.0507           1.33m
         6           1.0287           1.34m
         7           1.0059           1.33m
         8           0.9853           1.32m
         9           0.9715           1.31m
        10           0.9598           1.31m
        11           0.9430           1.31m
```

```
121         0.4085          5.13s
122         0.4068          4.40s
123         0.4037          3.66s
124         0.4007          2.93s
125         0.3983          2.19s
126         0.3952          1.46s
127         0.3915          0.73s
128         0.3889          0.00s
```

```
In [306]:
acc=accuracy_score(y1_pred, y1_test)
print('Tree Accuracy by hold-out evaluation: ',acc)
confusion_matrix(y1_test, y1_pred)
precision = precision_score(y1_test, y1_pred, average='binary')
print('Precision: %.3f' % precision)
recall = recall_score(y1_test, y1_pred, average='binary')
print('Recall: %.3f' % recall)
f1 = f1_score(y1_test, y1_pred, average='binary')
print('F1 score: %f' % f1)

Tree Accuracy by hold-out evaluation:  0.7974570505187957
Precision: 0.784
Recall: 0.961
F1 score: 0.863309
```

Accuracy: 0.79 Precision: 0.78 Recall: 0.86

Here, the accuracy obtained without performing PCA is higher.

6. SVM- Support Vector Machine

In machine learning, support-vector machines are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis.

Lastly, SVM was performed and following results were obtained.

SVM:

```
In [297]:
# SVM for diabetesMed

from sklearn.svm import LinearSVC

# by hold-out evaluation
#x1_train, x1_test, y1_train, y1_test = train_test_split(data_b_diab, y1_d, test_size=0.2)

clf = LinearSVC(random_state=0, tol=1e-5, max_iter=168)
#clf=SVC(kernel='linear', C=1E10) # C is large -> hard margin; C is small -> soft margin
clf=clf.fit(x1_train, y1_train)
y1_pred=clf.predict(x1_test)
acc=accuracy_score(y1_pred, y1_test)
print('Accuracy by hold-out evaluation: ',acc)
confusion_matrix(y1_test, y1_pred)
precision = precision_score(y1_test, y1_pred, average='binary')
print('Precision: %.3f' % precision)
recall = recall_score(y1_test, y1_pred, average='binary')
print('Recall: %.3f' % recall)
f1 = f1_score(y1_test, y1_pred, average='binary')
print('F1 score: %f' % f1)

Accuracy by hold-out evaluation:  1.0
Precision: 1.000
Recall: 1.000
F1 score: 1.000000
```

Accuracy: 1.00 Precision: 1.00 Recall: 1.00

SVM after PCA:

```
In [307]:
# SVM for diabetesMed after PCA

from sklearn.svm import LinearSVC

# by hold-out evaluation
x1_train, x1_test, y1_train, y1_test = train_test_split(PCAs, y1_dia, test_size=0.2)
#clf=SVC(kernel='linear', C=1E10) # C is large -> hard margin; C is small -> soft margin
clf = LinearSVC(random_state=0, tol=1e-5)
clf=clf.fit(x1_train, y1_train)
y1_pred=clf.predict(x1_test)
acc=accuracy_score(y1_pred, y1_test)
print('Accuracy by hold-out evaluation: ',acc)
confusion_matrix(y1_test, y1_pred)
precision = precision_score(y1_test, y1_pred, average='binary')
print('Precision: %.3f' % precision)
recall = recall_score(y1_test, y1_pred, average='binary')
print('Recall: %.3f' % recall)
f1 = f1_score(y1_test, y1_pred, average='binary')
print('F1 score: %f' % f1)

Accuracy by hold-out evaluation:  0.7951182173839089
Precision: 0.766
Recall: 1.000
F1 score: 0.867797
```

Accuracy: 0.79 Precision: 0.76 Recall: 1.00

The accuracy obtained without applying PCA is higher than that obtained after performing PCA.

5. Evaluations and Results

5.1. Evaluation Methods

The observed models were evaluated based on accuracy scores, precisions and recalls, to find the most significant algorithm for both the proposed problems.

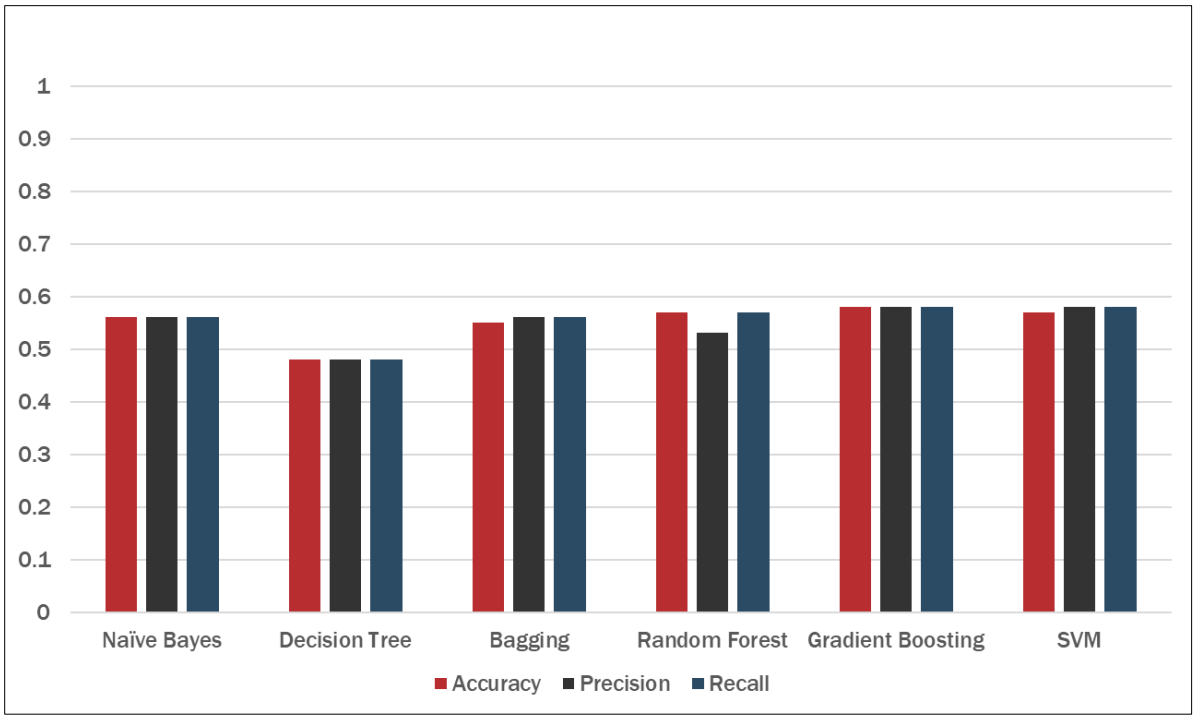
5.2. Results and Findings

A. Predicting Re-admissions

The following findings indicate which algorithm worked best based on its accuracy, precision and recall for predicting the re-admissions of the patients.

	Accuracy	Precision	Recall
Naïve Bayes	0.56	0.56	0.56
Decision Tree	0.48	0.48	0.48
Bagging	0.55	0.56	0.56
Random Forest	0.57	0.53	0.57
Gradient Boosting	0.58	0.58	0.58
SVM	0.57	0.58	0.58

Predicting Re-admissions:

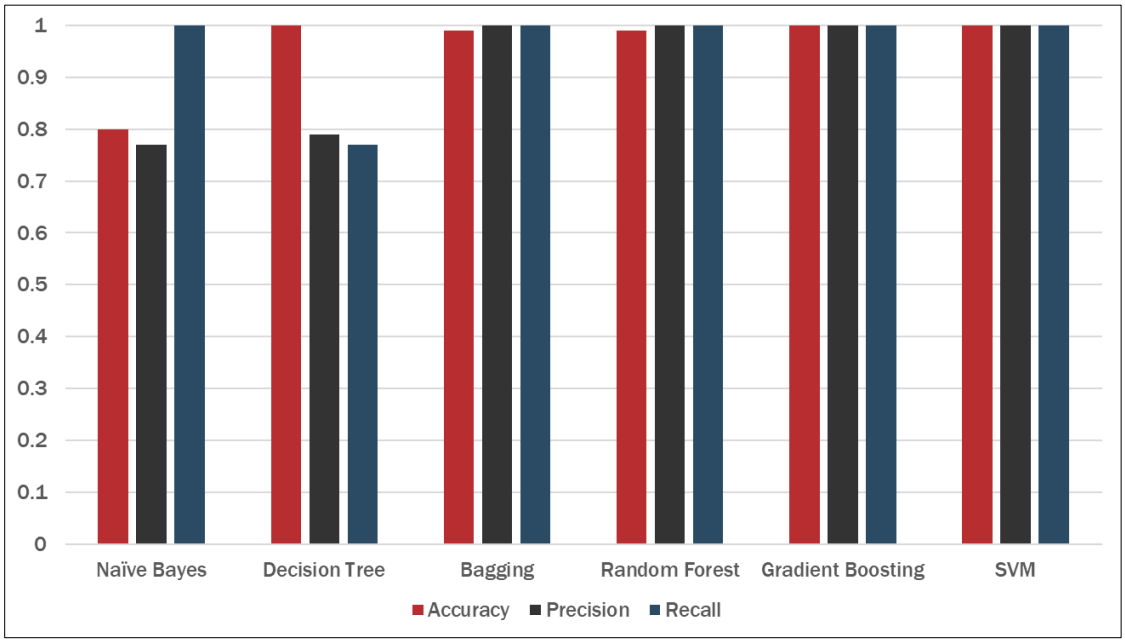


B. Predicting Diabetic's Medication Requirement

For predicting diabetic's medication requirements, various algorithms were observed. The following data shows comparison of all the models based on accuracy, precision and recall.

	Accuracy	Precision	Recall
Naïve Bayes	0.8	0.77	1
Decision Tree	1	0.79	0.77
Bagging	0.99	1	1
Random Forest	0.99	1	1
Gradient Boosting	1	1	1
SVM	1	1	1

Predicting Diabetic's Medication:



6. Over-fitting concerns

The best solution obtained after comparing all the algorithms to predict diabetic's medication requirement, seems to have too high accuracy and other evaluating factors. In order to confirm the trueness of it's working we checked for over-fitting issues.

We obtained the training accuracy and test accuracy for Gradient Boosting and Bagging for Decision Tree algorithm as well as for Naïve Bayes.

Train and test accuracy for Gradient Boosting with decision tree classifier:

```
print("Test Accuracy by Hold-out Eval:",accuracy_score(y1_pred,y1_test))
y1_pred_train=clf.predict(x1_train)
print("Train Accuracy by Hold-out Eval:",accuracy_score(y1_pred_train,y1_train))
confusion_matrix(y1_test, y1_pred)
precision = precision_score(y1_test, y1_pred, average='binary')
print('Precision: %.3f' % precision)
recall = recall_score(y1_test, y1_pred, average='binary')
print('Recall: %.3f' % recall)
f1 = f1_score(y1_test, y1_pred, average='binary')
print('F1 score: %f' % f1)
```

```
Test Accuracy by Hold-out Eval: 1.0
Train Accuracy by Hold-out Eval: 1.0
Precision: 1.000
Recall: 1.000
F1 score: 1.000000
```

Train and test accuracy for Bagging with decision tree classifier:

```
In [287]: # Bagging method for diabetesMed using Decision Tree classifier

#x1_train, x1_test, y1_train, y1_test = train_test_split(data_b_diab, y1_d, test_size=0.2)
tree = DecisionTreeClassifier()
bag = BaggingClassifier(tree, n_estimators=100, max_samples=0.8, random_state=1)
bag=bag.fit(x1_train, y1_train)
y1_pred=bag.predict(x1_test)
print("Test Accuracy by Hold-out Eval:",accuracy_score(y1_pred,y1_test))
y1_pred_train=bag.predict(x1_train)
print("Train Accuracy by Hold-out Eval:",accuracy_score(y1_pred_train,y1_train))
confusion_matrix(y1_test, y1_pred)
precision = precision_score(y1_test, y1_pred, average='binary')
print('Precision: %.3f' % precision)
recall = recall_score(y1_test, y1_pred, average='binary')
print('Recall: %.3f' % recall)
f1 = f1_score(y1_test, y1_pred, average='binary')
print('F1 score: %f' % f1)
```

```
Test Accuracy by Hold-out Eval: 0.9999574757611839
Train Accuracy by Hold-out Eval: 1.0
Precision: 1.000
Recall: 1.000
F1 score: 0.999968
```

Train and test accuracy for Naïve Bayes classifier:

```
In [284]:
# diabetesMed data split for balanced data
# Naive Bayes classification for diabetesMed in case of balanced data

x1_train, x1_test, y1_train, y1_test = train_test_split(data_final_dia, y1_dia, test_size=0.2)
clf = GaussianNB()
clf.fit(x_data, y_data)
y1_pred=clf.predict(x1_test)
print("Test Accuracy by Hold-out Eval:",accuracy_score(y1_pred,y1_test))
y1_pred_train=clf.predict(x1_train)
print("Train Accuracy by Hold-out Eval:",accuracy_score(y1_pred_train,y1_train))
confusion_matrix(y1_test, y1_pred)
precision = precision_score(y1_test, y1_pred, average='binary')
print('Precision: %.3f' % precision)
recall = recall_score(y1_test, y1_pred, average='binary')
print('Recall: %.3f' % recall)
f1 = f1_score(y1_test, y1_pred, average='binary')
print('F1 score: %f' % f1)

Test Accuracy by Hold-out Eval: 0.8063020921925498
Train Accuracy by Hold-out Eval: 0.8037485515027163
Precision: 0.776
Recall: 1.000
F1 score: 0.873665
```

	TEST ACCURACY	TRAIN ACCURACY	PRECISION	RECALL	F1 SCORE
BAGGING	0.99	1	1	1	0.99
GRADIENT BOOSTING	1	1	1	1	1
NAÏVE BAYES	0.80	0.80	0.77	1	0.87

From the results obtained, it appears that there is no over-fitting problem.

7. Conclusions and Future Work

6.1. Conclusions

After comparing the outcomes of all the models and algorithm, following conclusions were drawn. Highest accuracy obtained for predicting Diabetic's Medication Requirements is of Gradient Boosting and SVM.

	ACCURACY	PRECISION	RECALL
GRADIENT BOOSTING	1	1	1
SVM	1	1	1

In order to predict the re-admissions of patients in hospitals, the best algorithms obtained are Gradient Boosting and SVM.

	ACCURACY	PRECISION	RECALL
GRADIENT BOOSTING	0.58	0.58	0.58
SVM	0.57	0.58	0.58

6.2. Limitations

The best accuracy obtained for predicting re-admissions of patients in hospital is very low and hence not much reliable.

6.3. Potential Improvements or Future Work

The accuracy for predicting Re-admissions of patients in the hospital can be further improved by processing more advanced data mining techniques.

If provided with more precise data in terms of time period (month, day, year), time series analysis can be performed to determine the entire period the patient might be required to be under medication in the hospital.

The data can be further used to build a recommendation system to help doctors to determine the type of medications to be prescribed to the patients.

8. References

1. Hindawi BioMed Research International : https://www.hindawi.com/journals/bmri/2014/781670/#copyright
2. UCI Machine Learning Repository: http://archive.ics.uci.edu/ml/datasets/Diabetes+130-US+hospitals+for+years+19992008